МИНОБРНАУКИ РОССИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ**

**ВЫСШЕГО ОБРАЗОВАНИЯ**

**“ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”**

Факультет *компьютерных наук*

Кафедра *программирования и информационных технологий*

*Расширения для сервиса Trello по распределению загруженности*

*Курсовой проект*

09.03.02 *Информационные системы и технологии*

*Программная инженерия в информационных системах*

Обучающиеся \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*Д.И. Науменко, Н.С. Смолянинов 3 курс, д/о*

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*Х.А. Полещук, ассистент*

Воронеж 2019

**Содержание**

**Введение 3**

**1. Постановка задачи 4**

**2. Анализ 5**

**2.1. Анализ существующих решений 5**

**2.2. Анализ предметной области 6**

**2.3. Анализ задачи 6**

**2.4. Варианты использования 9**

**3. Дизайн 10**

**3.1. Архитектура 10**

**3.2. Диаграммы 13**

**4. Реализация 22**

**4.1. Средства реализации 22**

**4.2. Клиентская часть 23**

**4.3. Серверная часть 23**

**4.4. Развертывание 24**

**4.5. Тестирование 24**

**5. Интерфейс 25**

**5.1. Стартовая страница 25**

**5.2. Страница авторизации 26**

**5.3. Страница подтверждения 27**

**5.4. Страница выбора доски 28**

**5.5. Страница доски 28**

**5.6. Просмотр одного пользователя 29**

**5.7. Список всех пользователей 29**

**5.8. Результат распределения 30**

**Заключение 31**

**Список использованных источников 32**

**Приложения 33**

**Приложение 1. Техническое задание 33**

Введение

Trello – популярный бесплатный сервис для управления командными проектами. Аналогичен сервису Jira, но, в отличии от второго, свободно распространяемый. Количество пользователей в Trello уже к 2012 году составило более 1 миллиона пользователей. Сейчас этот сервис используется очень многими командами – как крупными проектами, так и небольшими стартапами.

В Trello присутствует удобный интерфейс для контроля заданий и отслеживания их статуса. Однако, несмотря на многие, в том числе и пользовательские, расширения, в Trello отсутствуют механизмы балансировки загруженности исполнителей в команде.

Расширения позволяют нам ввести понятие «Стори поинты». Стори поинты – это очки, с помощью которых исполнитель оценивает сколько времени он затратит на выполнение того или иного задания. Но, как мы и сказали ранее, нет никакого механизма, с помощью которого менеджер проекта просматривал бы количество стори поинтов каждого исполнителя. Если у одного исполнителя завал, а второй приходит только чаю попить, то кроме как долго и нудно покопавшись в заданиях этой проблемы не исправить.

Актуальность работы состоит в том, что время – ограниченный ресурс и в современном мире у человека его все меньше и меньше. Это приложение помогает сэкономить время на выполнении конкретной задачи.

1. Постановка задачи

**Цель –** создать сервис, позволяющий проектным менеджерам отслеживать, и распределять загруженность исполнителей на проектах.

**Сфера использования -** применяется в компаниях, которые пользуются Trello для распределения загруженности исполнителей

Для достижения цели были выдвинуты следующие задачи:

* Провести анализ требований к разрабатываемой системе.
* Провести проектирование приложения.
* Реализовать приложение, удовлетворяющее указанным требованиям, и описать процесс разработки и итоговый результат.

Более подробно задачи и требования описаны в техническом задании (Приложение 1).

1. Анализ

Так как реализации подобных задач уже существуют, то логично начать анализ с существующих решений.

* 1. Анализ существующих решений

**Clarizen.** Крупное приложение, предоставляющее много статистики, включая нагруженность, однако нагруженность присутствует только в демонстративном виде и советов по перераспределению не дается. Кроме того, приложение является платным.

**Paymo.** Другая альтернатива Trello с большим фокусом на автоматическое отслеживание времени, предоставляющее возможность бесплатного пользования одному пользователю. TrelloSuggest фокусируется на нагруженности, а следовательно на множестве пользователей, что и отличает его от Paymo.

**Gitscrum.** Данное веб-приложение предоставляет схожую с Trello функциональность, однако модуль расчета загруженности в эту функциональность не входит. К тому же, у Gitscrum отсутствует мобильное приложение.

**Blossom.** Приложение не предлагает возможностей TrelloSuggest, аналитический модуль фокусируется на рассчете производительности, не имеется мобильного приложения. Также Blossom не обладает бесплатной версией.

**Jira.** Предоставляет очень много разнообразных возможностей и услуг, в том числе аналитику и отчеты. Многое из этого направлено на производительность, но не связано с основным назначением TrelloSuggest.

Таким образом, функциональность TrelloSuggest представлена в небольшом числе приложений и, если представлена, то только частично. Кроме того, бесплатность Trello позволяет предоставить более высокий уровень доступности.

Так как Trello распространяется как в мобильной версии, так и в десктопной, было принято решение разработать приложение для Android и веб-интерфейс.

* 1. Анализ предметной области

Для корректного и полного отображения рабочей области Trello необходимо собрать информацию о:

* досках;
* списках;
* карточках;
* участниках;

Доска - проект, над которым работает пользователь. Содержит список списков.

Список – контейнер для карточек, группирующий их по выбранным пользователем критериям.

Карточка – единичное задание, которое может быть назначено одному или нескольким пользователям.

Участник – пользователь Trello, который является исполнителем на этой доске.

* 1. Анализ задачи

Задачу можно разбить на подзадачи:

* получение данных;
* отображение данных;
* реализация алгоритма распределения загруженности;

Рассмотрим задачу получения данных.

Для корректной работы нашего приложения нам необходимы сведения пользователей с их Trello-проектов. Чтобы получать эти данные самым логичным решением было выбрать официальный ресурс – Trello API. Там, имея «разрешение» пользователя, мы можем получить всю необходимую нам информацию.

«Разрешение» пользователя мы получаем путем аутентификации с помощью Trello OAuth. Этот сервис возвращает токен, с которым открывается доступ к приватным и командным данным пользователя.

Существует множество аналогов данной системы, и идея состоит в том, чтобы использовать их как ресурсы. Некоторые из них предоставляют свой API, а значит простые http-запросы могут решить нашу проблему нехватки данных, но этого недостаточно.

Чтобы Trello «узнавал» наше приложение – был получен приватный API Key. Каждый запрос данных сопровождается API Key и токеном пользователя.

Рассмотрим задачу отображения данных.

Для корректного и полного отображения доски достаточно ссылки на бэкграунд доски, названия доски и id доски (для того, чтобы можно было перейти внутрь и просматривать уже список списков).

Для корректного и полного отображения списка достаточно его названия и id (чтобы получить все карточки данного списка).

Для корректного и полного отображения карточки достаточно ее названия и количества стори поинтов.

Для корректного и полного отображения участников достаточно ссылки на их аватар, полного имени, имени пользователя и суммы всех стори поинтов со всех карточек пользователя.

Все решения по поводу пользовательского интерфейса принимались с упором на упрощение интерфейса. Чем более интуитивно-понятен интерфейс, тем он проще.

Рассмотрим задачу реализации алгоритма распределения загруженности.

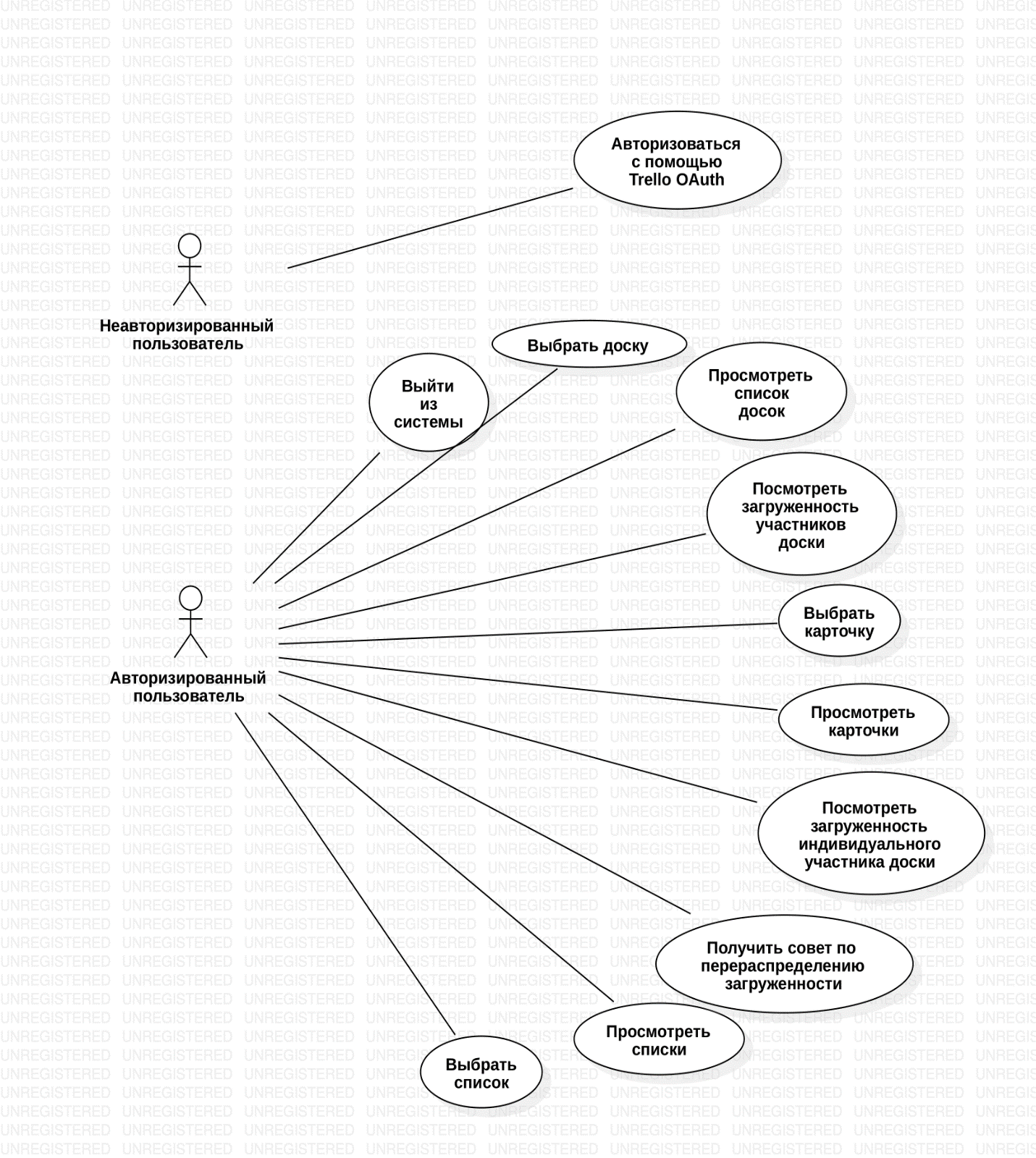
Алгоритм основан на сортировке и обмене данных в массиве массивов целых чисел. Целыми числами выступают стори поинты карточки. Вместо смены местами просто чисел, меняются объекты, содержащие в себе стори поинты и id карточки.

Алгоритм работает по принципу: отсортируй пользователей по количеству стори поинтов. Перемести задачу у самого загруженного самому свободному. Повторяй пока разница не окажется минимальной.

Алгоритм возвращает все перестановки карточек. Какая карточка, от какого пользователя забрали и какому пользователю передали.

* 1. Варианты использования

Конечная точка анализа – варианты использования, которые отображены на диаграмме прецедентов (Рисунок 1).

  
Рисунок 1 - Диаграмма прецедентов

1. Дизайн

По результатам анализа принимаются решения о дизайне системы. В этой главе представлены принятые решения и их обоснование.

* 1. Архитектура

В данный момент в современном мире распространены два типа архитектуры: монолитная и сервисная. Для того, чтобы выбрать тип, который будет реализовывать данная система были выявлены плюсы и минусы каждого из них:

* + Монолитная архитектура - одноярусное приложение, в котором компоненты объединены в одно приложение для одной платформы.

Плюсы:

* + - * простота тестирования;
      * простота развертки;
      * простота горизонтального масштабирования;
      * быстрота общения модулей;

Минусы:

* + - * Размер приложения замедляет его запуск.
      * Любое внесённое изменение требует перезапуска всего приложения.
      * Сложность масштабирования – разные модули обращаются к одним и тем же ресурсам.
      * Ошибка в одном из модулей останавливает весь процесс.
      * При смене инструментов (языка программирования) приходится детально перерабатывать все приложение.
  + Сервисная архитектура - подход разработки, при котором приложение строится как набор слабосвязанных модулей. Каждый такой отвечает за свою задачу и использует простой, строго определенный интерфейс для взаимодействия с другими сервисами.

Плюсы:

* + - * Могут быть развернуты на разных машинах, что ускоряет запуск.
      * Любое изменение влечет за собой перезапуск только одного конкретного сервиса.
      * Простота масштабирования – API-документация позволяет легко общаться сервисам и у каждого модуля могут быть свои отдельные ресурсы.
      * Некорректная работа одного сервиса не влечёт за собой падение всего приложения.
      * Легко комбинируются разные инструменты разработки в связи со строго определенным HTTP-взаимодействием.

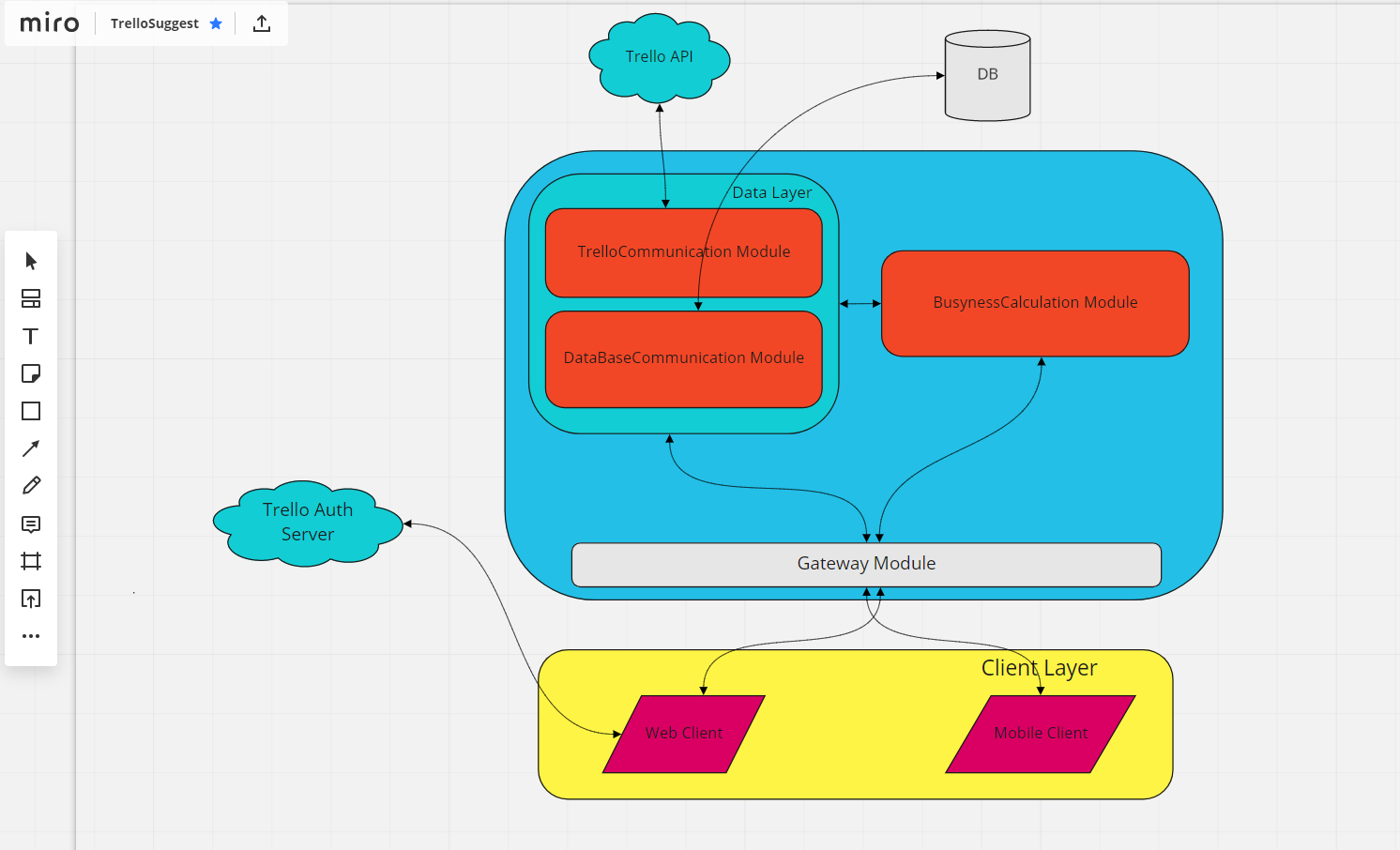
Минусы:

* + - * затрудненное End-To-End тестирование;
      * больше памяти, так как каждый сервис разворачивается в своем контейнере;
      * скорость общения модулей заметно ниже;

На основании всего вышеизложенного была выбрана монолитная архитектура. Главной причиной выбора монолитной архитектуры перед сервисной является размер приложения. Его сложно поделить на сервисы таким образом, чтобы не замедлить взаимосвязь его модулей.

Архитектура приложения приведена на рисунке 2.

Рисунок 2 – Архитектура приложения



* + - Web client – фронтенд данного приложения. Используется для взаимодействия с пользователями. Также на данном модуле производится аутентификация пользователя.
    - Mobile client – запланированный, но не реализованный мобильный клиент данного приложения.
    - Gateway Module – модуль, который отвечает за обработку HTTP-запросов к данной системе.
    - DatabaseCommunication module – модуль, который отвечает за взаимодействие данного приложения с базой данных.
    - TrelloCommunication module – модуль, который отвечает за взаимодействие данного приложения с Trello API.
    - BusynessCalculation module – модуль, который отвечает за распределение загруженности между исполнителями.
  1. Диаграммы

С помощью диаграмм можно описать систему/приложение с огромной подробностью.

Интерфейс пользователя TrelloSuggest организован следующим образом – после авторизации пользователь попадает на страницу досок. Оттуда он переходит на страницу отдельной доски, на которой отображены все списки и можно просмотреть пользователей (как всех, так и отдельного), также на этой странице можно запросить совет по распределению загруженности на всех списках или на одном конкретном.

Это отображено на последующих диаграммах состояния и активности.

Рисунок 2 – Диаграмма состояния

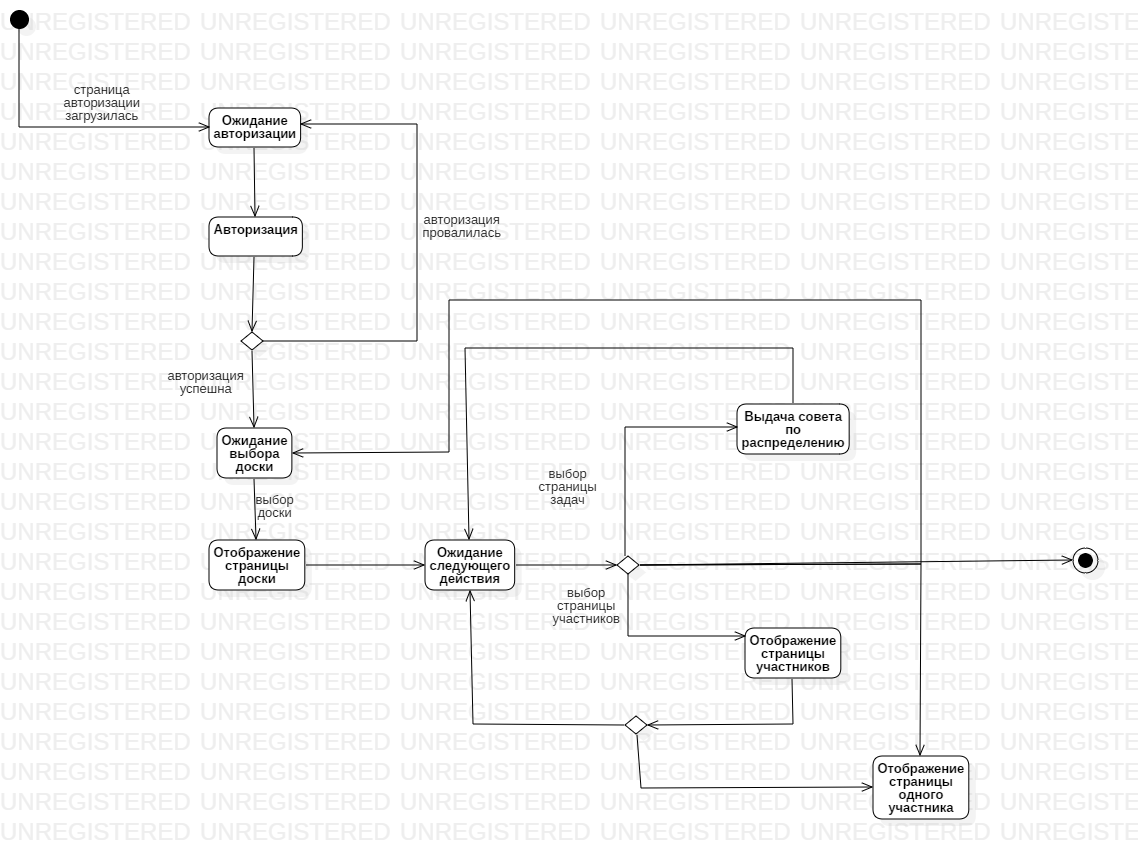
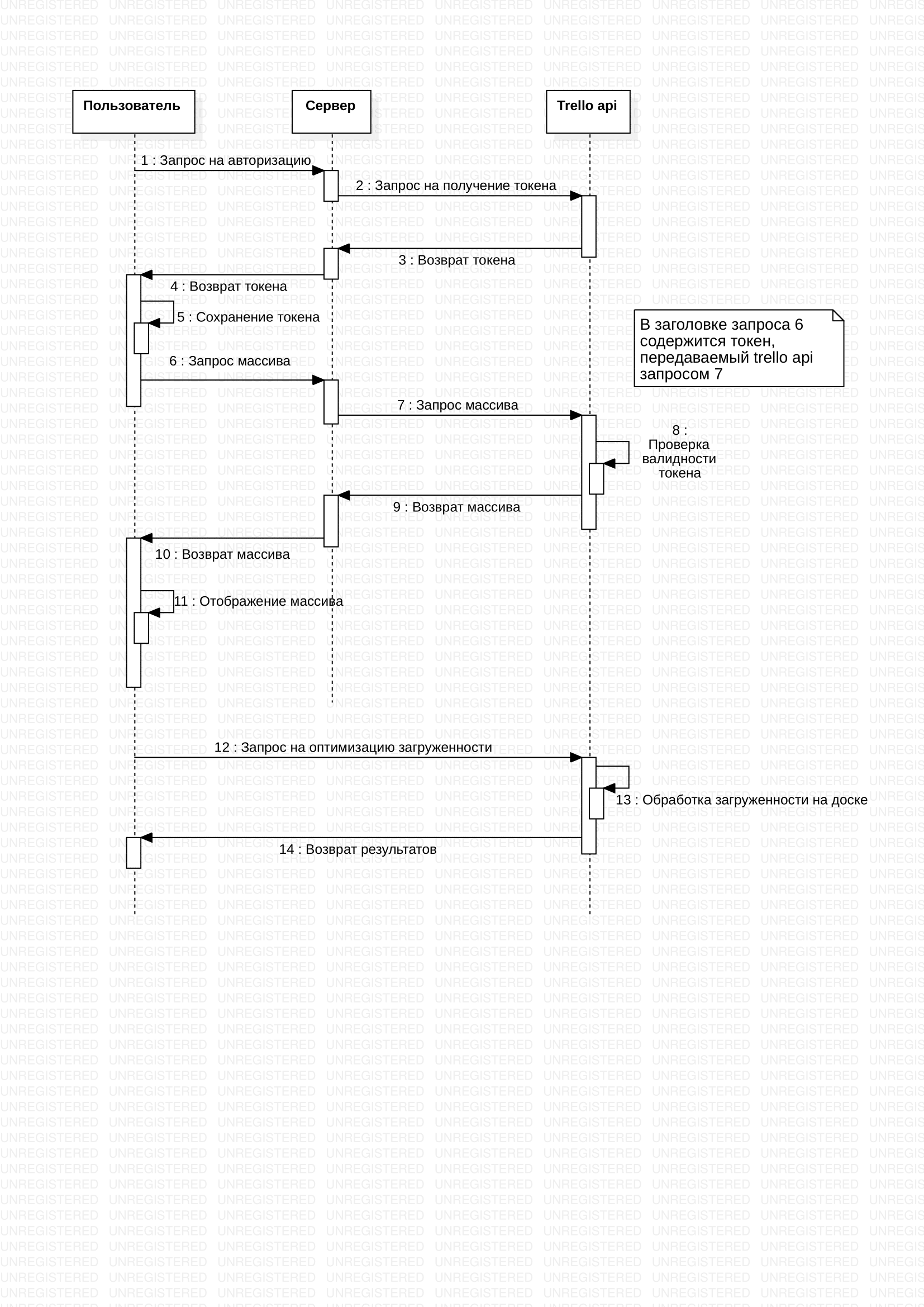


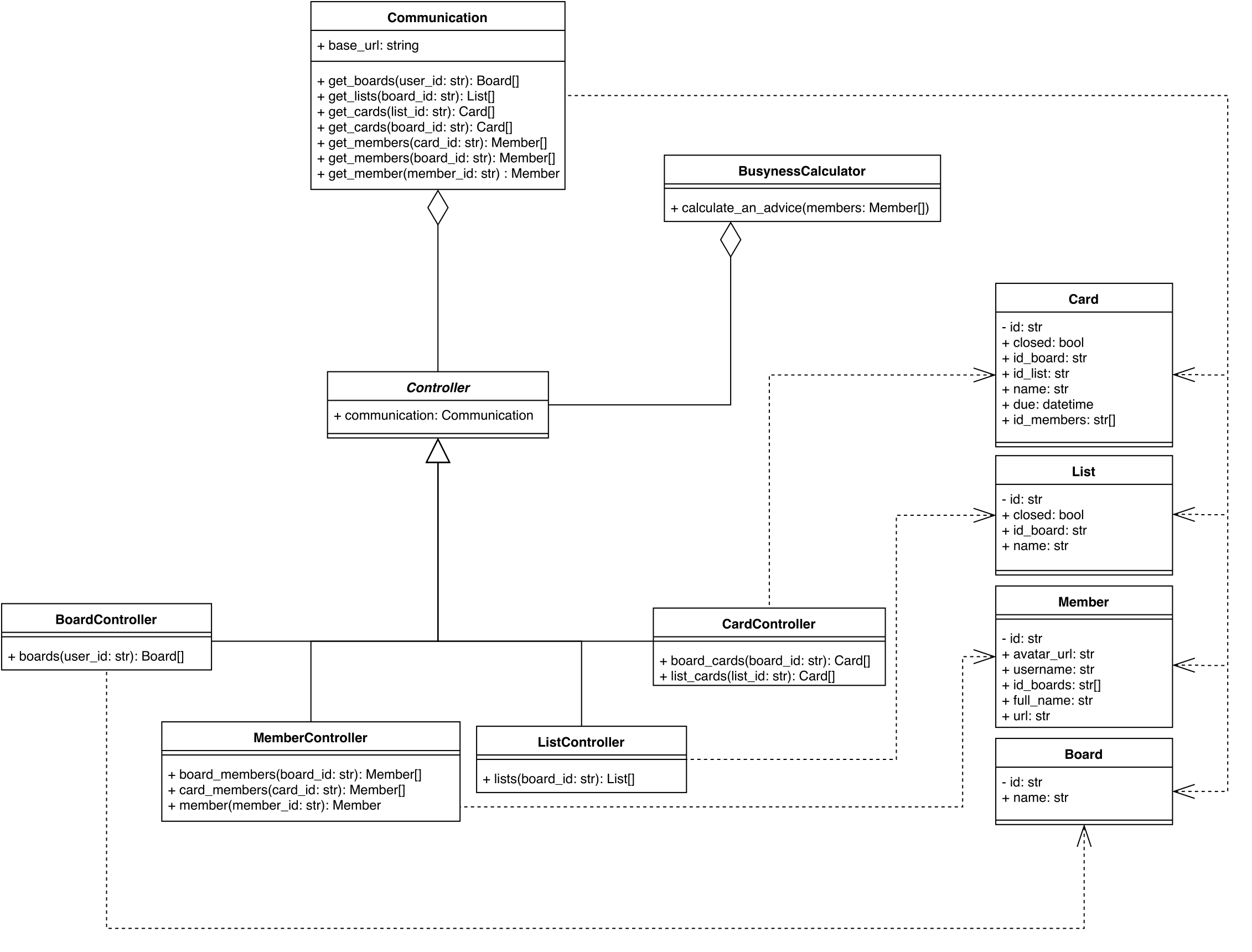
Рисунок 3 – Диаграмма активности



Далее представлена диаграмма последовательности основного пользовательского сценария аутентификации с последующим получением массива объектов.

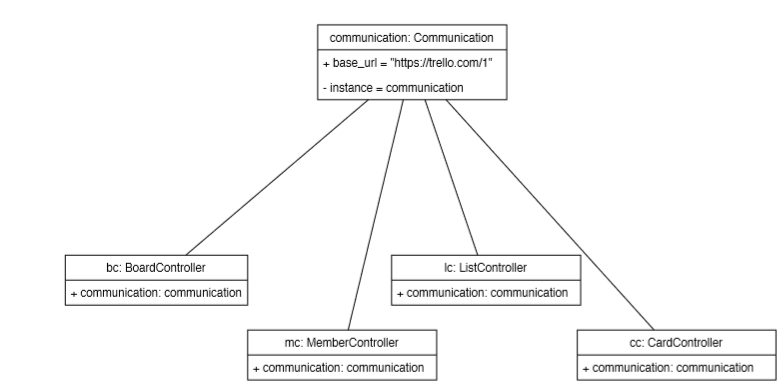
  
Рисунок 4 – Диаграмма последовательности

Далее представлена диаграмма классов, на которой показаны взаимодействия классов данной системы между собой.

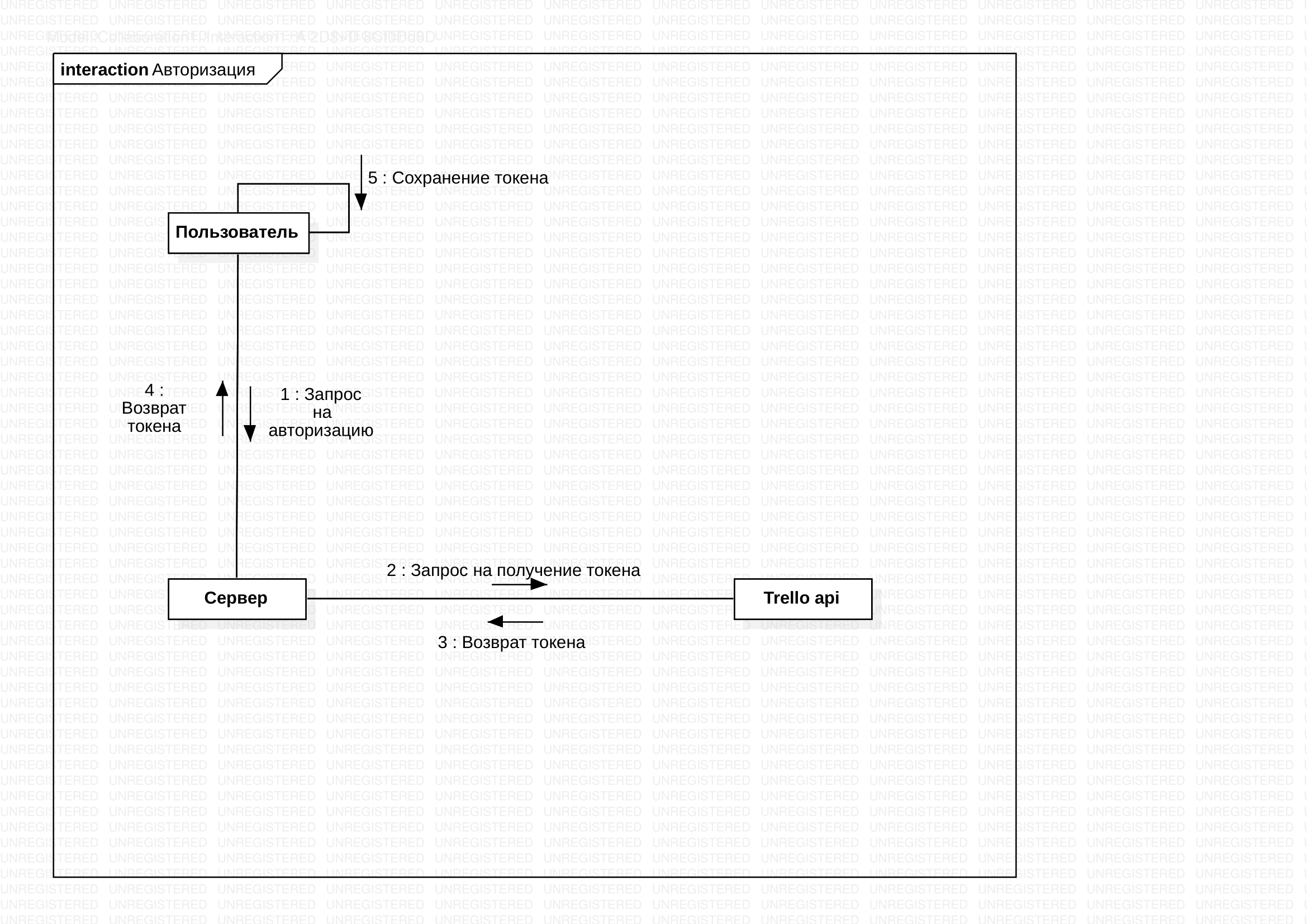
  
Рисунок 5 – Диаграмма классов

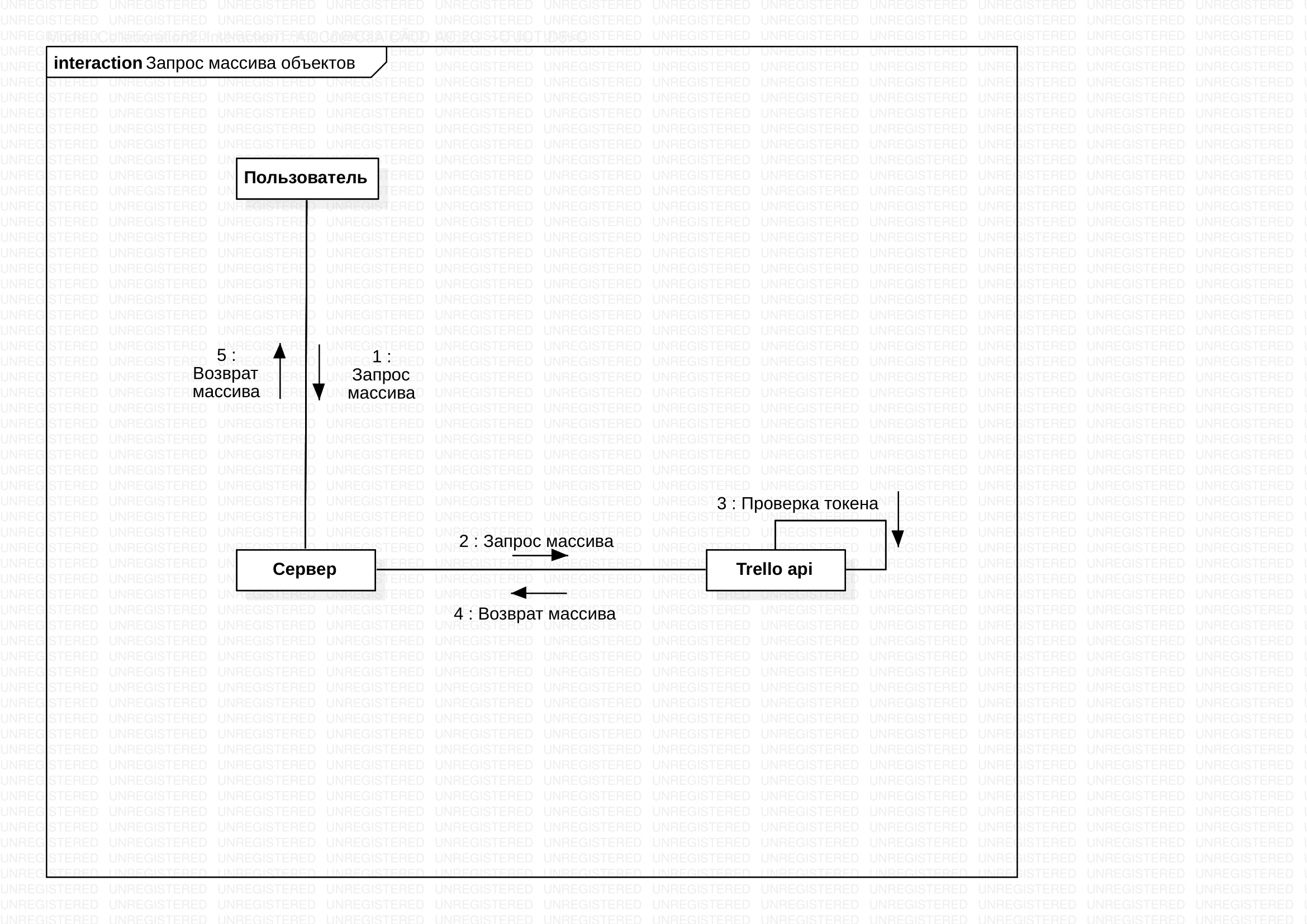
На рисунке 6 представлена диаграмма объектов, которая показывает собой конкретный случай инстанцирования классов типа Controller и Communication, чтобы показать, что у каждого объекта класса Controller хранится ссылка на один и тот же объект класса Communication.

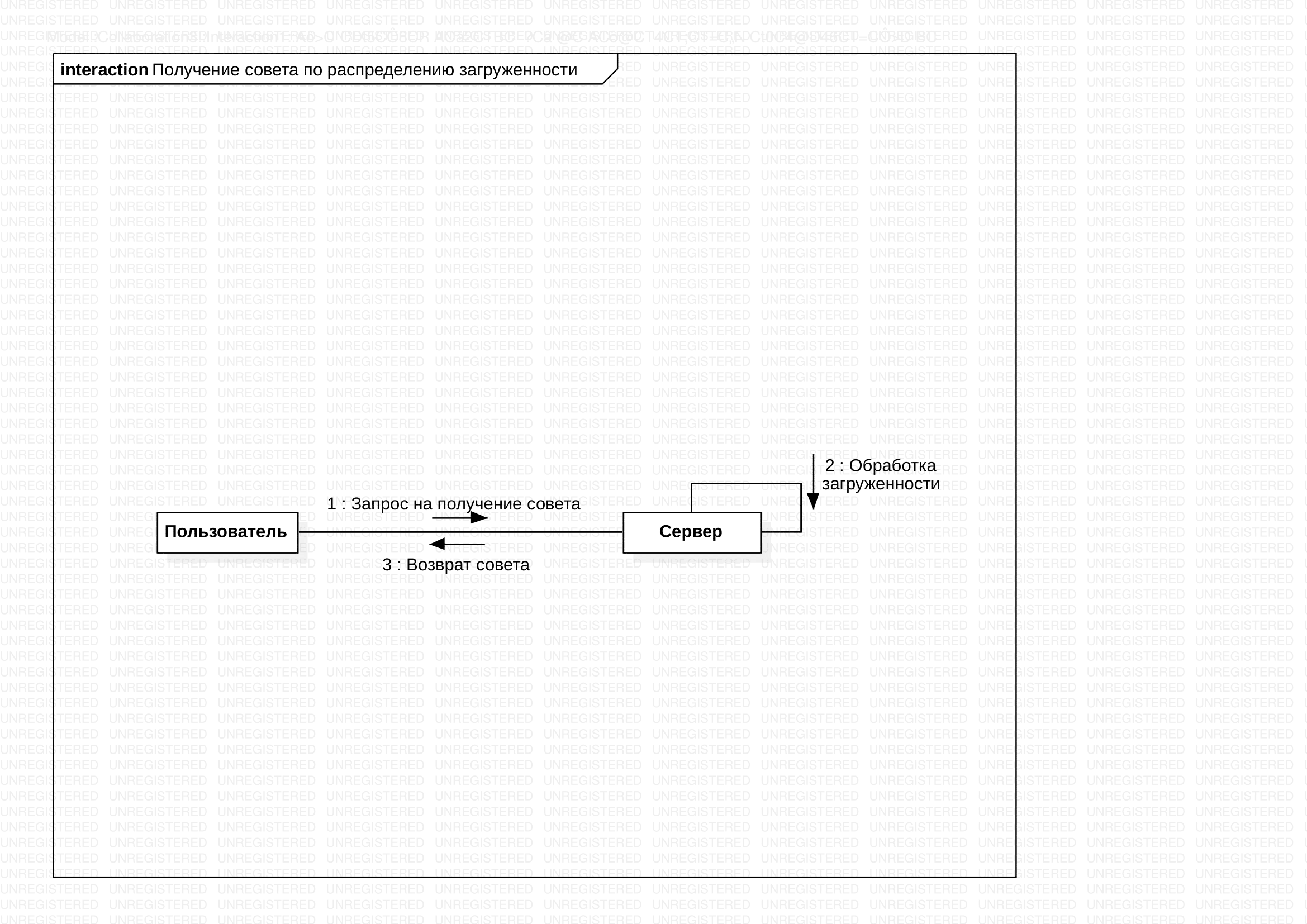
Рисунок 6 – Диаграмма объектов



Для более полного понимания взаимодействия модулей нашей системы как между собой, так и со сторонними модулями, приводятся диаграммы коммуникаций разных процессов.

  
Рисунок 7 – Диаграмма коммуникации процесса авторизации

  
Рисунок 8 – Диаграмма коммуникации процесса отображения списка объектов

  
Рисунок 9 – Диаграмма коммуникации процесса получения совета

На сервере разворачивается приложение, которое взаимодействует с Trello API. На этом же сервере располагается база данных PostgreSQL. Клиентская часть приложения представляет из себя веб-сайт и мобильное приложение. Взаимодействия между клиентской и серверной частью осуществляются посредством системы REST запросов.

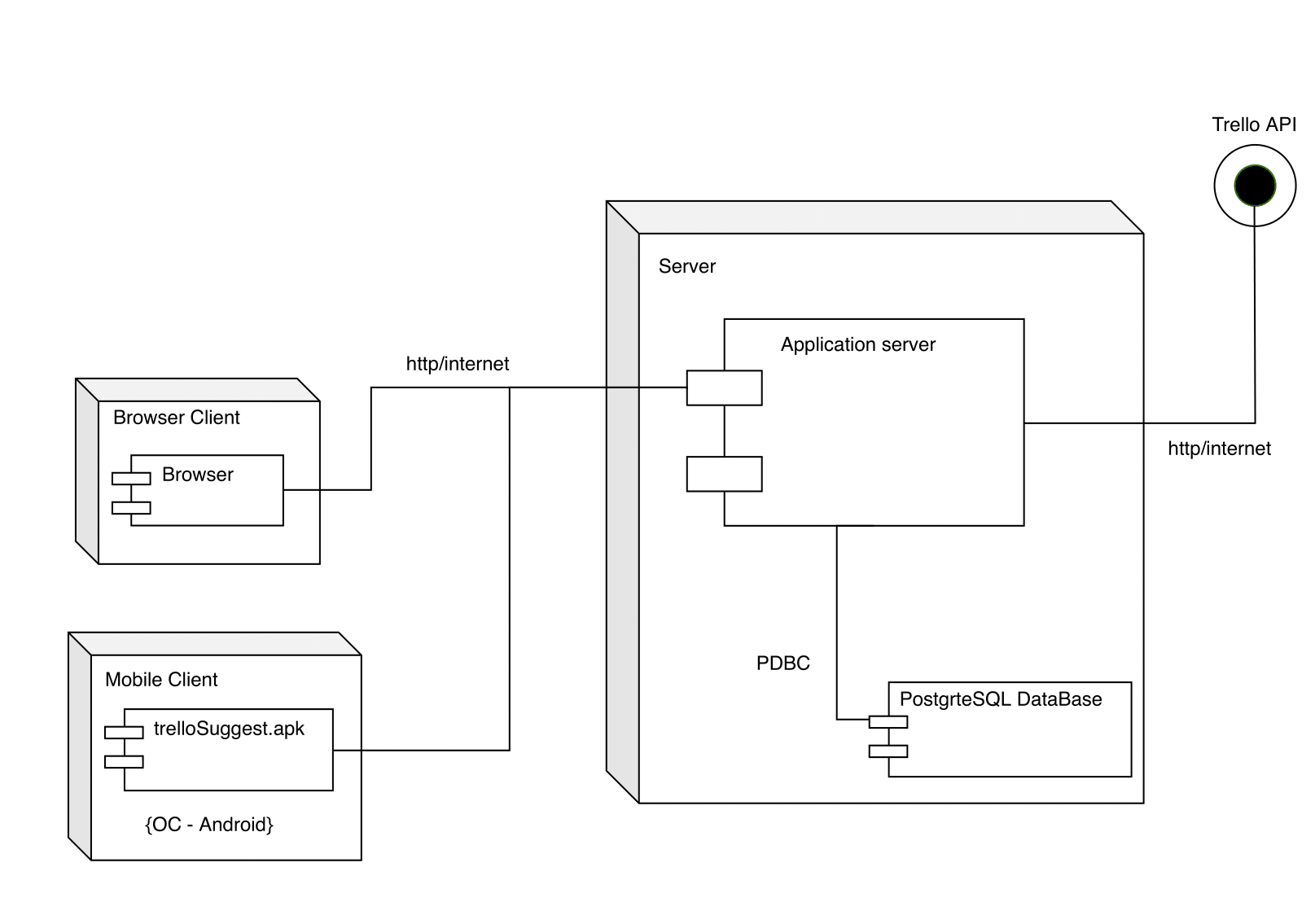
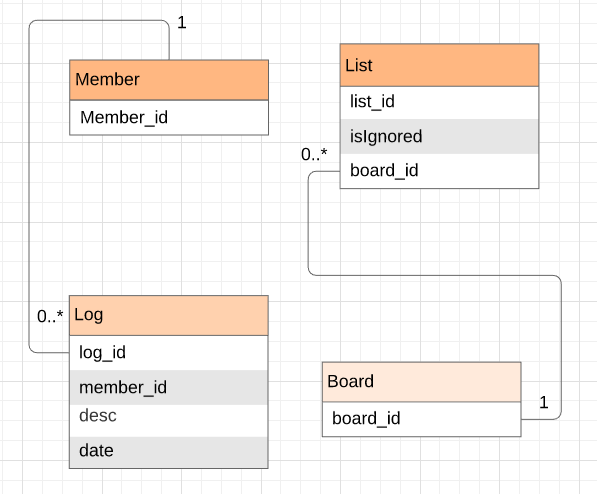
  
Рисунок 10 – Диаграмма развертывания

Рисунок 11 – ER-диаграмма



1. Реализация
   1. Средства реализации

В качестве средств реализации системы были выбраны следующие технологии:

* + - В качестве СУБД для всех сервисов была выбрана PostgreSQL. Выбор проводился между PostgreSQL, Oracle, MySQL, Sqlite. Oracle – слишком дорогая, Sqlite – слишком непроизводительна для использования на продакшн-уровне, MySQL – недавно была приобретена Oracle’ом, поэтому, опасаясь того, что она тоже станет дорогой, была выбрана PostgreSQL. Также, так как проект все-таки учебный, преследовалась цель научится.
    - Для разработки клиентской части приложения была выбрана платформа Angular версии 7 (на момент начала разработки – самой актуальной). Он позволяет создавать одностраничные веб-приложения с разделением данных и представления в соответствии со паттерном MVC. В Angular применяется двустороннее связывание: любые изменения в пользовательском интерфейсе сразу же отражаются на объектах приложения и наоборот. Платформа сама следит за событиями браузера, изменениями модели и действиями пользователя на странице, чтобы сразу обновлять нужные шаблоны.
    - Для реализации серверной части был использован язык Python и микрофреймворк Flask. Так как python – самый быстро-развивающийся язык в мире на данный момент.
  1. Клиентская часть

Клиентской частью приложения является Web Client.

Этот сервис отвечает за отображение данных и коммуникацию с клиентом.

Представляет из себя набор разных модулей, компонентов и сервисов.

Модули – классы, которые отвечают за конфигурацию работы компонентов и сервисов. Например – app.module.ts отвечает за импорт сторонних библиотек и утилит, которые используются на сервисе. Особенность в том, что он не только импортит их, но и экспортит. Таким образом – не нужно в каждом отдельном файле дописывать импорт какой-либо утилиты, если этот импорт есть в app.module.ts. App-routing-module.ts отвечает за то какой компонент должен в данный момент отображаться на основе url, на котором находится клиент.

Компоненты – классы, разбитые на 3 файла: шаблон (html), стили (scss), typescript class (ts).

Сервисы – классы, в которые выносится общая для нескольких компонент логика. Например – backend.service.ts. Сервис отвечает за общение клиентского приложения с бэкендом с помощью http-запросов (rest api), передавая JSON.

* 1. Серверная часть

На серверной части реализована логика, подразумевающая логирование действий пользователя, хранение данных о досках пользователей, конкретнее – список списков, у которых есть определенное поле, отвечающее за то, будет ли список рассматриваться при распределении нагрузки или нет.

В качестве библиотеки для работы с базой данных используется peewee – она предоставляет возможность использовать ORM-запросы и поддерживает postgresql без дополнительных настроек.

Flask предоставляет простоту, гибкость и аккуратность в работе, позволяя пользователю самому выбирать, как реализовать те или иные вещи, а Django предоставляет пакет «все включено. Flask был выбран из соображений получения большего опыта и возможности для обучения.

* 1. Развертывание

Приложение было развернуто с помощью утилиты Digital Ocean по ip-адресу 68.183.218.223.

С помощью winc-connection были установлены необходимые библиотеки и база данных. Также была установлена и проинициализирована база данных.

* 1. Тестирование

В данной работе проводилось 3 вида тестирования: юнит-тесты, регрессионное тестирование и системное тестирование.

Юнит тестам был подвергнут алгоритм распределения загруженности. Он проверялся на:

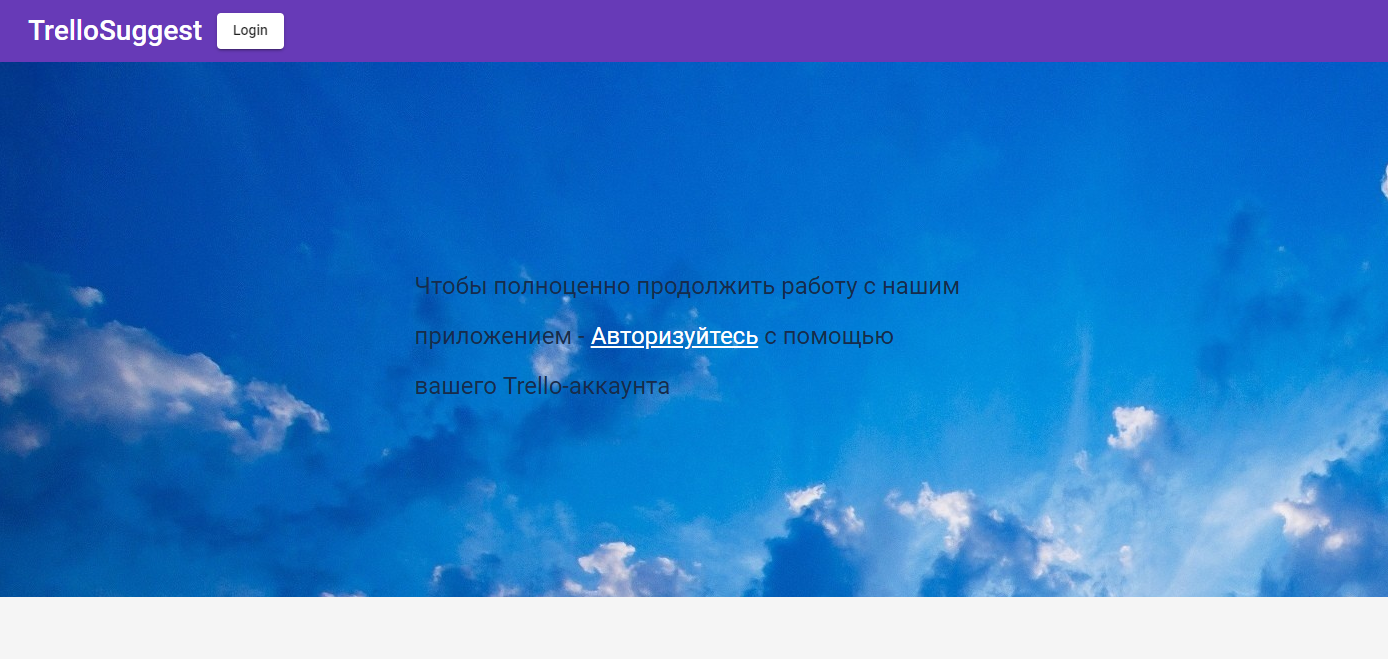
* Финальный разброс между минимальным и максимальным уменьшается.
* Не изменяет данные (не удаляет и не добавляет новые сущности).

Регрессионное тестирование – при подключении к проекту базы данных было обнаружено, что некоторые данные некорректно отображаются. Оказалось, проблема в конфликте данных из разных источников. Проблема была устранена путем вывода данных после их маппинга на бэкенде.

Системное тестирование было выполнено на реальной доске Trello с прогонами всех пользовательских сценариев. По итогам тестирования были выявлены недочеты отображения данных на фронтенде. Недочеты были исправлены путем написания новых стилей.

1. Интерфейс
   1. Стартовая страница

Рисунок 11 – Стартовая страница приложения



Чтобы начать работу с данным приложением необходима авторизация, так как в ином случае приложение не будет иметь доступ к приватным данным пользователя.

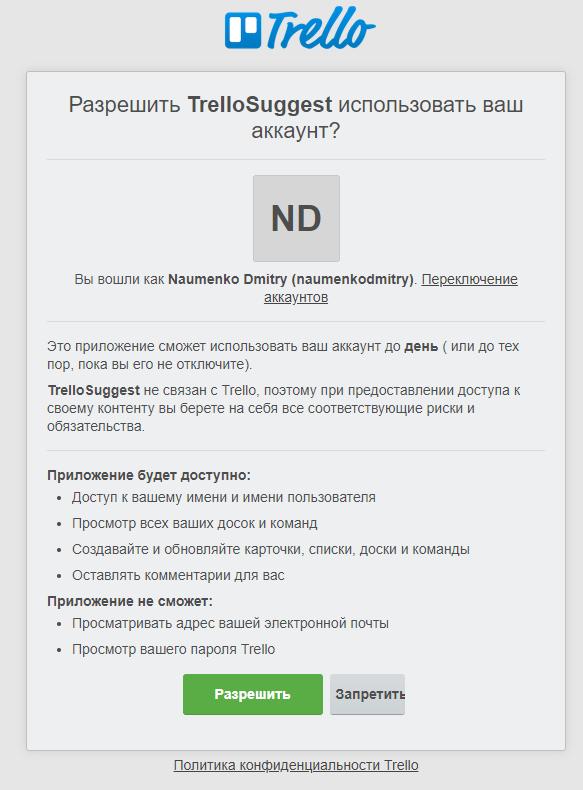
* 1. Страница авторизации

Рисунок 12 – Форма авторизации Trello



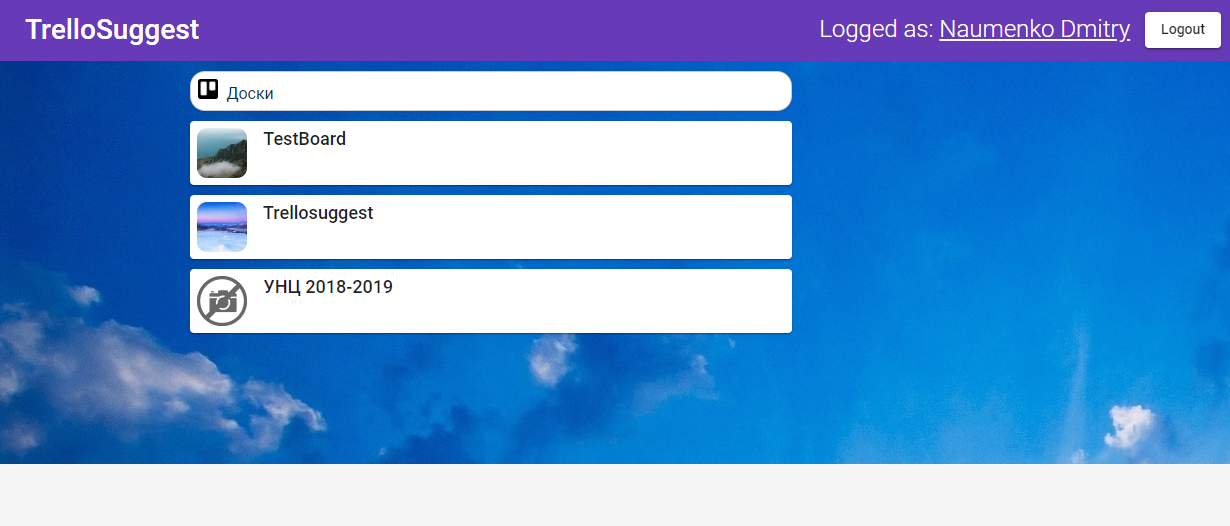
* 1. Страница подтверждения

Рисунок 13 – Форма подтверждения прав приложения



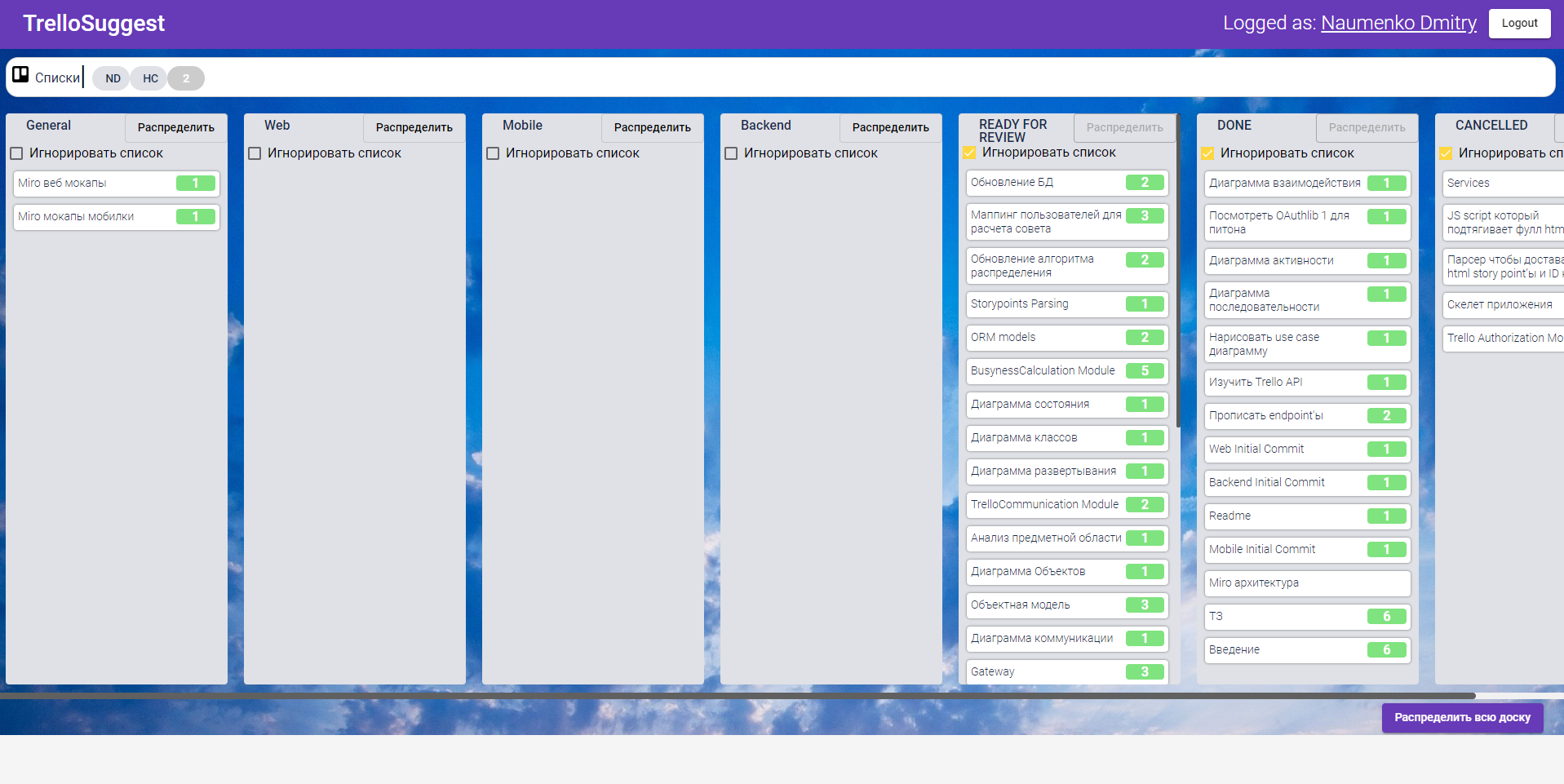
* 1. Страница выбора доски

Рисунок 14 – Страница выбора доски



* 1. Страница доски

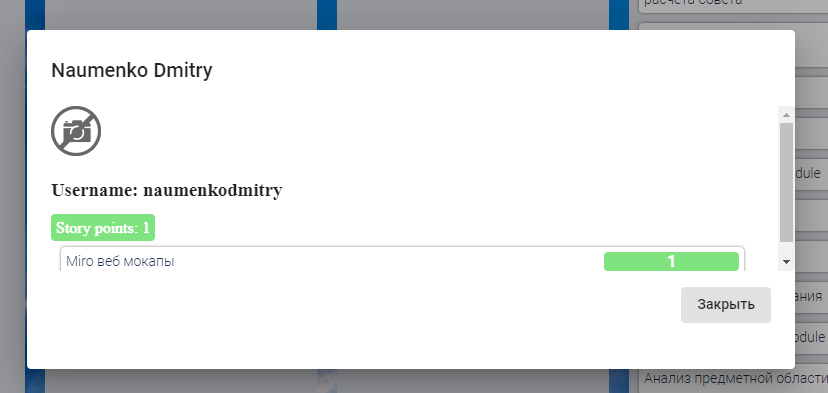
Рисунок 15 – Страница доски



На данной странице можно просмотреть пользователей доски (как всех, так и конкретного), отметить списки, которые не будут учитываться при распределении загруженности, распределить загруженность отдельного списка или всей доски сразу. Также можно попасть обратно на страничку с выбором доски путем нажатия на надпись “TrelloSuggest” или выйти из системы, нажав кнопку “Logout”.

* 1. Просмотр одного пользователя

Рисунок 16 – Информация о пользователе



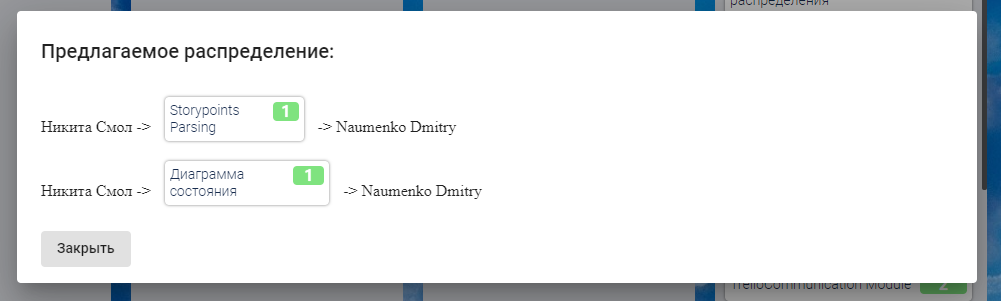
* 1. Список всех пользователей

Рисунок 17 – Список всех пользователей



* 1. Результат распределения

Рисунок 18 – Результат распределения



Заключение

В ходе выполнения работы были проанализированы существующие решения сервисов для распределения загруженности, в результате анализа был сделан вывод, что ни одно из них полностью не соответствует желаемому результату.

Также в ходе выполнения было выполнено проектирование системы. Составлены диаграммы классов, объектов, последовательностей, состояний, коммуникаций, прецедентов, активности, развертывания и модульная схема.

Результатом выполнения работы является система, позволяющая распределить загруженность, пользуясь сервисом Trello.

Список использованных источников

* + - 1. Jeremy Wilken Angular in action / Jeremy Wilken – MANNING, 2018. – 320 с.
      2. Paul Barry Head First Python A Brain-Friendly Guide / Paul Barry – O’REILLY, 2017. – 624 c.
      3. Trello API Documentation [[https://developers.trello.com](https://developers.trello.com/)]

Приложения

Приложение 1. Техническое задание

1. **Общие сведения**

1.1 Наименование приложения:

TrelloSuggest

1.2 Сфера применения:

Возможно применение в компаниях, которые пользуются Trello для распределения загруженности исполнителей

1.3 Сроки:

1. Срок начала: 27.03.2019
2. Срок сдачи: 04.06.2019 08:00
3. **Термины и определения**

2.1 Общие термины

1. ТЗ - Техническое задание (текущий документ)
2. TBD – To Be Defined. Секция в ТЗ, которая должна быть определена позже.
3. FAQ – Frequently Asked Questions. Часто задаваемые вопросы.
4. ВИ – Вариант Использования или Use Case.
5. ДВИ – Диаграмма Вариантов Использования или Use Case Diagram.

2.2 Бизнес термины

1. Доска - отдельный проект в Trello, в котором содержаться карточки, отсортированные по спискам
2. Список - тот самый список из пункта (1)
3. Карточка - компонент, в котором есть краткое описание задачи для исполнения, информация о том, кто выполняет эту задачу, сроках и стори поинтах
4. Стори поинты - очки, которые исполнители присваивают заданию, чтобы оценить его сложность (примерное количество времени, которое будет затрачено на реализацию)

2.3 Технические термины

1. ОС - операционная система
2. БД - база данных
3. ПО - программное обеспечение
4. Ендпоинт - URL, по которому может быть выполнен http-зпарос
5. Модуль - часть приложения, которая выполняет узконаправленный набор функций
6. **Назначение и цели создания**

3.1 Назначение документа

В настоящем документе приводится набор требований к ПО, необходимых для реализации.

При реализации необходимо выполнить работы в объёме, указанном в настоящем ТЗ.

3.2 Цели создания ПО:

3.2.1 С точки зрения создателей:

Сформировать клиентскую базу для оказания дополнительных услуг.

3.2.2 С точки зрения клиента:

* Упростить процесс распределения нагрузки на исполнителей с помощью приложения для контроля задач Trello.
* Уменьшить время, затрачиваемое на распределение нагрузки.

1. **Характеристика объектов автоматизации**

4.1 Объект автоматизации:

Распределение стори поинтов в системе управления задачами Trello.

4.2 Условия использования объекта автоматизации:

Главное условие - взаимодействие нескольких исполнителей на одном проекте (в иных случаях использование не оправдано).

1. **Требования к системе**

5.1 Функциональные требования

5.1.1 Основные функциональные возможности:

* Просмотр всех своих досок
* Просмотр всех карточек с доски
* Просмотр всех листов с доски
* Просмотр загруженности каждого человека на доске
* Просмотр загруженности всех пользователей доски
* Получение советов по оптимальному распределению нагрузки на пользователей

5.3 Нефункциональные требования

5.3.1 Требования к безопасности

1. Система аутентификации - OAuth
2. Все секретные данные шифруются
3. Все ендпоинты на запись – защищены

5.3.2 Требования к надежности

1. При ожидаемом поведении пользователя багов/падений встречаться не должно
2. Логирование падений
3. Система должна работать для следующих браузеров последних версий: Mozilla Firefox, Google Chrome, Opera.
4. Система должна стабильно работать с глубиной истории не менее чем в 3 месяца
5. Система должна стабильно работать при 50 одновременно подключенных пользователях.
6. **Модульная схема**

